Spring 1-1-2012

Scale Invariant Object Recognition Using Cortical Computational Models and a Robotic Platform

Danny Voils
Portland State University

Scale Invariant Object Recognition Using Cortical Computational Models and a Robotic Platform

by

Danny Voils

A thesis submitted in partial fulfillment of the
requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Thesis Committee:
Dan Hammerstrom, Chair
Marek Perkowski
Christof Teuscher

Portland State University

Abstract

This paper proposes an end-to-end, scale invariant, visual object recognition system, composed of computational components that mimic the cortex in the brain. The system uses a two stage process. The first stage is a filter that extracts scale invariant features from the visual field. The second stage uses inference based spacio-temporal analysis of these features to identify objects in the visual field.

The proposed model combines Numenta's Hierarchical Temporal Memory (HTM), with HMAX developed by MIT's Brain and Cognitive Science Department.

While these two biologically inspired paradigms are based on what is known about the visual cortex, HTM and HMAX tackle the overall object recognition problem from different directions. Image pyramid based methods like HMAX make explicit use of scale, but have no sense of time. HTM, on the other hand, only indirectly tackles scale, but makes explicit use of time. By combining HTM and HMAX, both scale and time are addressed.

In this paper, I show that HTM and HMAX can be combined to make a complete cortex inspired object recognition model that explicitly uses both scale and time to recognize objects in temporal sequences of images. Additionally, through experimentation, I examine several variations of HMAX and its interaction with HTM in an object recognition task.

Contents

Abstract i

List of Tables v

List of Figures vi

1 Introduction 1

1.1 Research Background and Motivation 1

1.2 Thesis Organization 2

2 Vision and the Brain 4

2.1 Why Study the Brain? 4

2.2 Invariant Features and Object Recognition 5

2.3 Scale, Object Recognition and Current Work 6

2.4 A brief tour of the Visual Cortex 7

3 Hierarchical Temporal Memory 10

3.1 Bayesian Networks 10

3.2 Bayesian Inference and the Visual Cortex 12

3.3 The HTM Network 13

3.3.1 The HTM Node 14

3.3.2 Spatial Pooling 15

3.3.3 Temporal Pooler 18

3.4 Hidden Markov Models and HTM 21

3.4.1 Training an HTM Network 23
3.5 Using K-Means for Spatial Pooling 26
3.6 HTM and Scale Invariance 26

4 The Object Recognition Filter **29**
4.1 Introduction 29
4.2 Convolution and the Pyramid Transform 29
4.3 Scale Invariant Features 32
4.4 HMAX, Revisiting the Visual Cortex 34
4.4.1 S1 Level 34
4.4.2 C1 Level 35
4.4.3 S2 Level 36
4.4.4 C2 Level 37
4.5 Serre's Contribution 37
4.5.1 S1 Improvements 37
4.5.2 C1 Level 39
4.5.3 S2 Level Improvements 40
4.5.4 C2 Level 40
4.6 Intelligent Signal Processing 40

5 Experiments **42**
5.1 Cortex Inspired Object Recognition 42
5.2 The Experiment Software 43
5.2.1 The SRV1 Robot 43
5.2.2 HMAX Filter 44
5.3 The Experiments 45

5.3.1 Experiment 1 47
5.3.2 Experiment 2 47
5.3.3 Experiment 3 48

6 Results and Conclusions **49**
6.1 Results 49
6.1.1 HTM level 1, Spatial Pooler's view 49
6.1.2 Object Recognition 51
6.1.3 Analysis 53
6.2 Conclusions 54
6.3 Future Directions 55

References **57**

Appendix A Software **61**
A.1 ORFilter 61
A.1.1 Filter Layers 61
A.1.2 Auxiliary Procedures 62
A.2 HTMZ1 62
A.2.1 Node 63
A.2.2 TPool 64
A.2.3 SPool 65
A.2.4 Link 65

List of Tables

5.1 Experiments 47

5.2 Image Scales used in full HMAX implementation 48

6.1 Mean Variance, HTM level 1 51

6.2 Experiment 1, Confusion Matrix 51

6.3 Experiment 1, Object Recognition Results 52

6.4 Experiment 2, Confusion Matrix 52

6.5 Experiment 2, Object Recognition Results 52

6.6 Experiment 3, Confusion Matrix 53

6.7 Experiment 3, Object Recognition Results 53

6.8 HTM Memory Usage by Hierarchical Layer 54

List of Figures

2.1 A simplified schematic of the Visual Cortex 8

3.1 A simple Bayesian Network 11
3.2 A 3 level hierarchy of nodes in an HTM network 14
3.3 HTM Node 15
3.4 An example of image segmentation 16
3.5 Vorinoi Tesselation 17
3.6 Temporal Grouping Example 19
3.7 Hidden Markov Model 22
3.8 Temporal Grouping and Markov Trellis 23
3.9 Training a 3 level HTM hierarchy 25
3.10 Effects of scaling on image features 28

4.1 Convolution 30
4.2 Image Pyramid 32
4.3 Simplified Riesenhuber and Poggio Model with two scales and two rotations. 35
4.4 Feature Orientation -vs- Scale 36
4.5 Simplified Serre model with two rotations and two scales 38
4.6 First derivative of Gaussian (top), Gabor (bottom) 39

5.1 Bio-Inspired Object Recognition 42
5.2 Robot Vision System 43
5.3 SRV1 Open Source Robot 44

5.4 Test Subjects 46
5.5 The three experiments; illustrating increasingly sophisticated signal processing blocks. 47
6.1 HTM Level 1, spatial pooler output 50
6.2 Biologically Inspired Computing System 56

Chapter 1

Introduction

1.1 Research Background and Motivation

We live in an age of amazing computing machines. Today we have computers that can simulate nuclear explosions, beat us at games like Chess and Jeopardy, and even drive our cars. Despite these impressive advancements, many tasks that a human can easily perform remain difficult, if not impossible, for today's machines.

Just walking around a kitchen or folding laundry challenges even the most sophisticated machines and computing models [7]. If computers and machines are going to help us in every day life, they will need to contend with all the uncertainties of our dynamic, and often chaotic, world.

Today's computers face other challenges as well. Every year since the invention of the integrated circuit, computers have become smaller, faster and cheaper, following the trajectory of Gordon Moore's well known premise, "Moore's Law." His prediction, that the number of transistors on a chip roughly doubles every 18 months, has been a driving force behind the modern high-tech economy. However, as devices drop below 10 nanometres it's unclear whether the traditional Boolean computing paradigm will continue to hold. A new paradigm, based on current understanding of the brain's structure, could hold the key to computing in this "non-boolean" world.

Recent discoveries in the field of Computational Neuroscience have yielded models of vision that are inspired by the cortex in the brain. Of particular interest is the role of scale invariance in vision. How is the visual system in the brain able

to instantly recognize things, even at a distance?

Although current computational models of vision have shown promising results, they do not use a completely cortical paradigm. Instead, they use a hybrid modeling approach, where the cortical model is assisted by traditional machine learning or signal processing components.

This thesis explores the origins of scale invariance in vision by using computational models inspired by the cortex. To accomplish this, a visual object recognition system that exclusively uses cortex-inspired computational models as components is created by combining HTM, developed by Nuementa Corporation with HMAX, developed at MIT's Brain and Cognitive Science Department. Three experiments are then used to examine how several different HMAX configurations interact with HTM to perform scale-invariant object recognition.

1.2 Thesis Organization

This Thesis is organized as follows:

Chapter 2 introduces concepts in biological vision. It provides a motivation for studying the brain and vision. The concept of image features are introduced. Finally, an introduction to the main elements of the visual cortex are discussed.

In Chapter 3, provides detailed background and analysis of Numenta's Hierarchical Temporal Memory (HTM). An introduction to Bayesian computational networks is provided before diving into the connections between the visual cortex and Bayesian Inference. The components of a HTM are discussed in detail, including both spatial and temporal components. An analysis of the mathematical aspects of HTM and its connection to Vector Quantization and Hidden Markov Models is provided.

Chapter 4 talks in detail about the HMAX model. It discusses the basics of convolution, and its importance in both traditional and biologically inspired image processing before introducing the image pyramid. It also details the history and current state of the art of HMAX before introducing the idea of Intelligent Signal Processing.

Chapter 5 the explains the three experiments performed in this thesis.

Chapter 6 provides an analysis of the experimental results along with final conclusions. Future work is discussed in this chapter, detailing improvements to the model and motivation for further study.

The appendix gives details on the software that was used to perform the experiments.

Chapter 2

Vision and the Brain

2.1 Why Study the Brain?

Nature has had millions of years to evolve the human brain. It contains over 100 billion parallel neural processing elements, weighs 1300 grams, and consumes only 25 watts of power[28]. Given the brain's almost magical ability to interact intelligently with the real world, it's not unreasonable to view the brain as the most advanced computing device known.

In fact, from the early days of computers, researchers have drawn parallels between computers and the brain. Early computers were called electronic brains, and computing pioneers like Turing and Von Neumann thought of the brain as a computing machine [35, 33]. Recently, the National Academy of Engineering has even made Reverse Engineering the Brain one of its grand challenges of the 21st century [23].

The brain is highly scalable. It can store, analyze, and create inferences from data almost instantly. The brain learns largely from the bottom up, rather than top down, as in traditional machine learning. Therefore, using the brain as a model for computing makes sense.

Most of the human brain is dedicated to processing visual information. In addition to being our most dominant sense, the ability to quickly recognize objects and others is an integral part of our relationships, our ability to perform tasks, and even our survival.

Therefore, it's not surprising that the visual cortex has been the subject of

intense study for many years. In fact, we now know more about the visual cortex than any other part of the brain. This knowledge has inspired new computational models of vision.

2.2 Invariant Features and Object Recognition

In the vast majority of vision based tasks, an essential step is recognizing objects in a scene. *Object recognition* is the process of associating *features* extracted from an image or sequence of images with their respective *objects*. Objects are anything of interest in a scene, such as a vase, a cat, or a person.

The object recognition problem becomes more complex when the observer, or the objects being observed, are moving in relation to each other. Observers recognize that objects are moving because they notice differences in scale and translation. Scale refers to the relative size of an object, while translation refers to lateral movement in the field of view.

Changes in *scale* are proportional to distance. In other words, close objects take up more of the visual field and appear larger, while more distant objects shrink in the visual field and appear smaller. Similarly, changes in *translation* are caused by lateral motions through time.

The visual cortex filters out features that remain relatively unchanged even during scale and translational transformations. Features that have this property are said to be *invariant* to such changes. By defining invariant features, the visual cortex can recognize objects in a dynamic environment where the observer and objects being observed may be in motion.

2.3 Scale, Object Recognition and Current Work

Adding the dimension of scale has two effects on the feature extraction task. First, object features can be represented at different scales, and thus the number of features that can be extracted increases.

Current research in image processing provides a number of ways to deal with the expanding feature base. Fergus [5] used Maximum Likelihood to estimate the parameters of important regions and scales to form a family of probability distributions in a Bayesian manner.

A similar approach that uses probability, but is not Bayesian, is described by Leibe et al. [16] and uses Harris interest points [10] in a voting scheme over a spatially fixed probability distribution of learned features and image patches. This is then extended by creating a third dimension of voting space using scale transformations.

Another method described by Lowe [17] proposes using second-closest neighbor features from multiple training images of the same object. Here, the closest neighbor of a correct match is closer than the closest incorrect match.

Second, incorporating multiple scales can reveal more details about an image. For example, Kadir [14] showed that saliency can be defined across both scale and feature space by using localized entropy. This can help point out important features in an image, and lead to faster object recognition.

However, an open question remains, how does the visual cortex handle scaling? Recently, advances in brain imaging techniques combined with increasingly sophisticated computational theories and an exponential scaling of computing power have come together to give researchers an ever more detailed understanding of how the brain solves scaling.

Early studies of the cat brain show that multi-scale processing is performed by the visual cortex [13]. Psychological studies show humans have the ability to recognize objects at multiple scales even after viewing the object only once. It is believed that the cortex does this by scaling receptive field properties to preferred spatial wavelength [32]. These discoveries have inspired researchers to develop computational models which explicitly use multi-scale processing of images.

However, these models do not directly address how multi-scale visual representations are processed over time. Likewise, very little is known how the brain uses scale and time together to help facilitate things like navigation.

2.4 A brief tour of the Visual Cortex

Before diving into the computational models, it's worthwhile to take a quick look at the biology. Studies show that the visual cortex, like the rest of the brain, is roughly segmented into functional areas. In vision, these areas have been shown to be arranged in a hierarchical manner with *early* visual processing functionally closest to the retina. Information travels from the retina to higher structures in the hierarchy where a combination of feed forward and feed back mechanisms are in place as illustrated in figure 2.1.

In early vision, the visual field is segmented into regions or receptive fields. The size of these receptive fields is smallest in early processing stages, and grows as one moves up into higher regions. Physically, cells are arranged in a laminar sheet with alternating layers dominated by two cell types: complex cells and simple cells. Simple cells are sensitive to specific orientations of spatial gratings. Complex cells tend to pool information from simple cells, thereby building invariant representations [12].

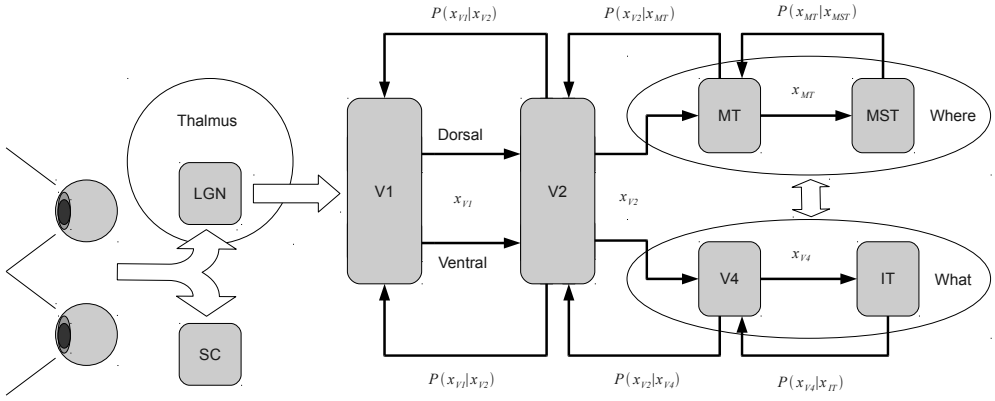

Figure 2.1: A simplified schematic of the Visual Cortex

Mountcastle was the first to theorize that the Cortex was arranged in a regular columnar structure [21, 20]. Vertical columns of cells that share the same modality extend through the laminar layers of Cortex, and represent the modular nature of the brain.

This modular structure, combined with short range horizontal connections and heavy use of hierarchy could be the key to the brain's ability to manage vast degrees of neural complexity. The predominate connections in the brain are to near neighbors. Fewer connections exist between distant structures. This organization may be what makes the brain highly scalable. That is, its function can be improved merely by adding more cortical column modules to the system.

The lowest stages of vision processing are believed to act like a multistage filter. In fact, even the retina is known to have some filtering properties. Images are sensed by the eye and encoded by the retina which conveys visual information to the lateral geniculate nucleus (LGN) in the Thalamus. From here, projections continue into various areas of the visual cortex, with the most predominate being to V1. Connections in the Cortex tend to be recurrent, and in fact, there are more

backward projections than forward projections between LGN and V1.

In V1, visual information is processed in a way similar to Gabor filtering. In addition, there are strong recurrent connections between V1 and V2. The V2 area is organized similar to V1, with regions tuned to specific rotations and shapes. Receptive field sizes are much larger in V2, however, and it has the ability to detect curves and illusory contours [22]. These areas are also involved with calculating binocular disparity for use in depth perception. From this point, information flows in two main processing paths with the ventral and dorsal streams dominating.

The dorsal path, thought to be mainly involved with motion information, has strong connections with the Medial Temporal (MT) lobe which is responsible for processing the movements and direction of objects within the field of view. This region is thought to perform optical flow calculations needed to process global image motion in situations where an observer moves through a landscape. It also controls eye movements, [24] as well as computing the trajectory of moving objects in the field of view [34].

Current theories suggest that the ventral path is used in object recognition. Invariant recognition of shape and 3D orientation has been shown in V4, which is strongly connected to early visual processing [11] as well as higher areas like the Inferotemporal Gyrus (IT) which has been shown to have shape selectivity over large receptive fields. This area has also been shown to respond to Fourier image components within a range of frequencies with a great deal of invariance [31].

Chapter 3

Hierarchical Temporal Memory

The real world is full of uncertainty. Machines that work in real environments must contend with noisy and unreliable sensors, unexpected events and an ever-changing environment. To help manage all this uncertainty, we can use ideas from probability theory. Recently, ideas from probability and statistics, such as Bayesian analysis, has been applied to problems in the field of Artificial Intelligence and Machine learning. Hierarchical Temporal Memory or HTM is one such construction. This paper focuses on the first generation HTM algorithm which is referred to as Zeta 1. The HTM model is borrows ideas from Bayesian Networks.

3.1 Bayesian Networks

Bayesian belief networks are directed acyclic graphs used to represent joint probability distributions between random state variables. Variables are modelled using graphical nodes while the edges define conditional probabilities between the variables.

Each node contains information about the joint probability of nodes it is connected to called a conditional probability table. In this way, the Bayesian network acts as a type of memory where the joint probabilities or *beliefs* are stored in individual nodes, and new information called *messages* are passed up the chain. Messages can go down the chain as well. These two classes of messages shown in Figure 3.1, and are called π and λ respectively.

Bayesian networks get their name from Bayes' Theorem, which was first coined

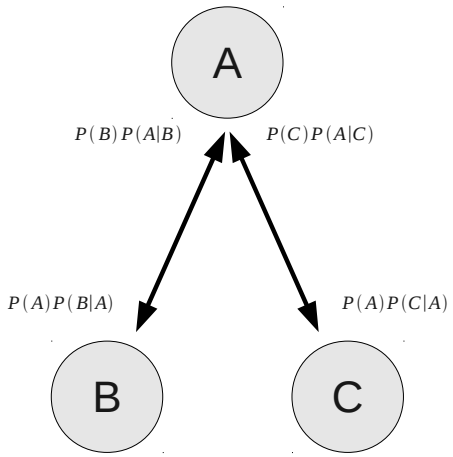

Figure 3.1: A simple Bayesian Network

by the 18th century English mathematician and Presbyterian minister, Thomas Bayes. The theorem describes a way to calculate so-called inverse conditional probabilities, injecting "beliefs" with the *prior* probability $P(B)$. Bayes' Law has been called the "most important equation in machine learning" [18].

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{3.1}$$

If the conditional probability of event A given event B is defined as $P(A|B)$, then Bayes theorem calculates the *inverse* probability $P(B|A)$.

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)} \tag{3.2}$$

The conditional probability $P(A|B)$ is called the *likelihood*, and is the hypothetical probability that an event, in this case, A will take on a particular value

or have a specific outcome, based on past observations of A and B. The term $P(B)$ is called the prior distribution. Bayes' rule states the posterior distribution is calculated by combining the likelihood with the prior probability.

After Bayes' death, the theorem lay in obscurity before being rediscovered by Pierre Simon Laplace as a form of probabilistic inductive reasoning. After Laplace, the theorem was "nearly obliterated by battling theorists" [19] before being rediscovered again in the mid 20th century.

Bayesian networks provide a method for reasoning about uncertainty. They provide a compact and expressive way to represent joint probability distributions. According to recent research, they can also be used to model how the brain works.

3.2 Bayesian Inference and the Visual Cortex

Regions of the brain are highly interconnected as shown in Figure 2.1. In addition to the dorsal and ventral paths, direct connections between various brain regions exist. These pathways or *projections* have been shown to go both forward and backward along processing stages in the visual cortex.

Lee and Mumford [15] were among the first to propose that probabilistic calculations performed in the visual cortex can be modelled using Bayesian Inference. They proposed a probability model that "captured the interaction between multiple cortical areas."

For example, the calculation for the ventral pathway is,

$$P(x_0, x_{V1}, x_{V2}, x_{V4}, x_{IT}) = P(x_0|x_{V1})P(x_0|x_{V2})P(x_0|x_{V4})P(x_0|x_{IT}) \tag{3.3}$$

where x_0 is the input image, x_{V1} is the output of V1, x_{V2} is the output of V2, etc.

Lee and Mumford showed that object recognition can be modelled as a belief update process where messages are sent between hierarchical processing regions of the visual cortex.

George and Hawkins [6] later adapted Bayesian belief propagation as part of a general computational framework based on the visual cortex.

3.3 The HTM Network

Hierarchical Temporal Memory is a learning algorithm developed by Numenta Inc. It uses a hierarchical framework of processing nodes that perform learning and computation. As the name suggests, HTM is a type of distributed memory. It consists of one or more networks of processing nodes connected using hierarchical layers.

The nodes contain memory elements, and are analogous to nodes in Bayesian belief networks. The hierarchy is arranged in a pyramidal structure as shown in Figure 3.2 with the leaf nodes at the bottom collecting input, while the highest nodes in the hierarchy make the final decision about the likely cause of the input. Parent nodes in higher levels collect information from multiple children in lower hierarchical levels. New information from the leaf nodes is processed, and passed up the hierarchical chain.

The HTM paradigm makes explicit use of time. Time is used to draw temporal connections between incoming streams of information. These correlations, or *coincidences,* are then combined and relayed up to higher levels where the process is repeated.

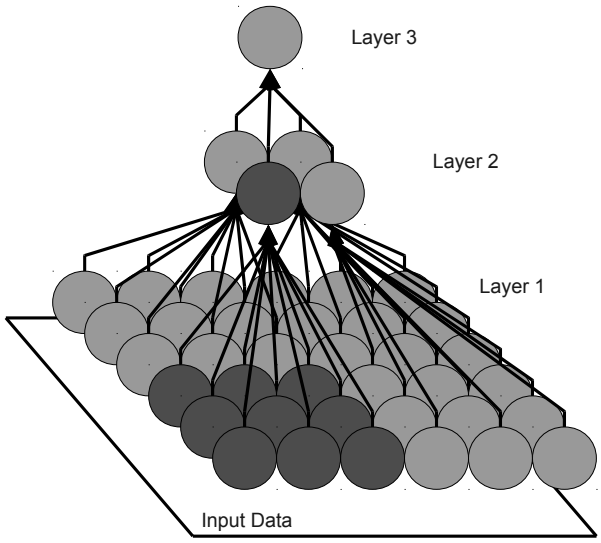

Figure 3.2: A 3 level hierarchy of nodes in an HTM network

3.3.1 The HTM Node

An HTM Node is made up of two parts connected in a pipeline configuration as shown in Figure 3.3. The spatial pooler has multiple inputs, and the nature of these inputs depends on the node's placement in the hierarchy. If the node is on the bottom, this means that it must act as an interface for data coming into the network.

Furthermore, each node has a receptive field, which is a term borrowed from neurobiology. The receptive field is defined as the input region for a sensory neuron. Here it defines what portion of the input space is allocated to a specific HTM node.

Figure 3.4 shows an example of a typical image segmentation task. Here each segment represents a receptive field that would be used by an HTM node at the

bottom of the hierarchy.

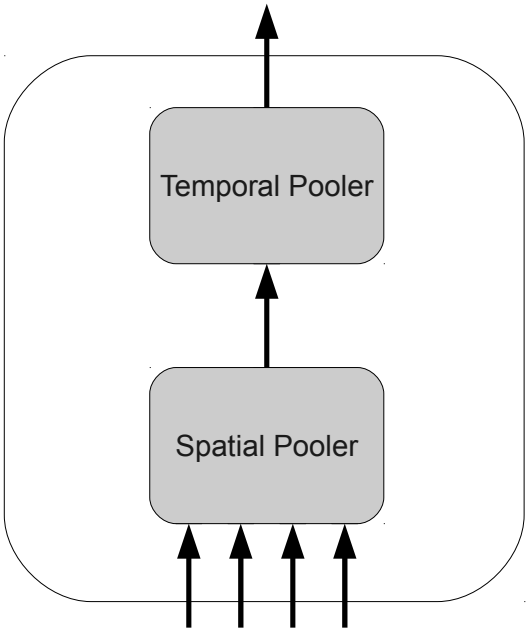

Figure 3.3: HTM Node

The idea of a receptive field in HTM is illustrated in Figure 3.2, where the dark colored nodes in the lowest layer feed into the node in the middle layer. Receptive fields subdivide the input space and may or may not overlap.

3.3.2 Spatial Pooling

Spatial pooling performs a type of competitive learning called vector quantization. Vector quantization segments the input space into discrete regions, or Vorinoi sets, as shown in Figure 3.5. In the case of the spatial pooler, the N dimensional input, represents either a piece of input data, or outputs from the set of child nodes.

Inputs are compared to each vector-valued entry in a *code book*. The comparison

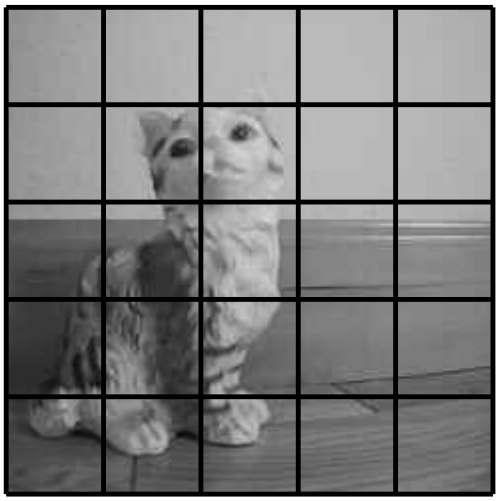

Figure 3.4: An example of image segmentation

used is some form of geometric distance, where the input and all the code book entries are N dimensional vectors. The code book entry with the minimum distance is the winner. For example, in the real case, the code book entry R_j with the minimum distance is defined as,

$$R_j = \{\mathbf{x} \in \mathbb{R}^N | j = argmin_{k=1...n} ||\mathbf{x} - \mathbf{c_k}||\} \tag{3.4}$$

where $\mathbf{x}$ is the input vector, and the code book, $\mathbf{c} = \{\mathbf{c_1}, \mathbf{c_2}, .., \mathbf{c_k}, .., \mathbf{c_n}\}$ contains n centers. The method used to calculate geometric distances depends on the type of data. For example, if the data is in integer form, Manhattan distance, or L1, Norm makes sense. This is defined as,

$$d_1(\mathbf{x_1}, \mathbf{x_2}) = \sum_{k=1}^{n} |x_{1k} - x_{2k}|. \tag{3.5}$$

A special case of Manhattan distance that is used on binary data is Hamming distance, where the distance between two binary vectors is the number of bits that are different between them. For real valued data, it makes sense to use Euclidean distance. This is defined as,

$$d_2(\mathbf{x_1}, \mathbf{x_2}) = \sqrt{\sum_{k=1}^{n} |x_{1k} - x_{2k}|^2}. \tag{3.6}$$

In all three cases $\mathbf{x_1}$ and $\mathbf{x_2}$ are N dimensional vectors.

This process is a type of cluster analysis where the code book entries are quantization centers. During the learning phase, quantization centers are created by adding entries to the code book.

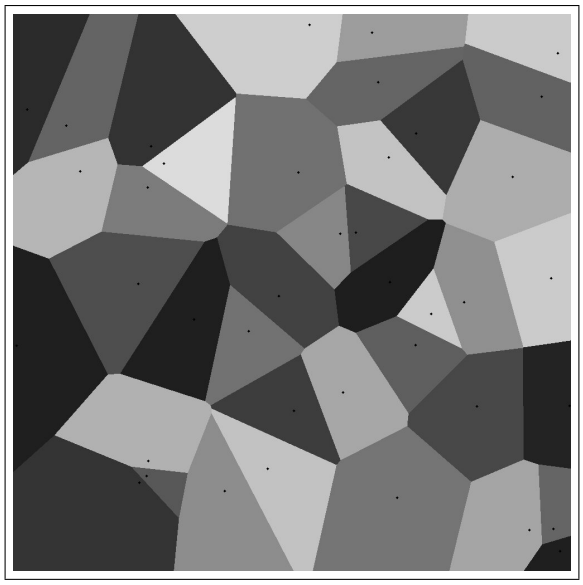

Figure 3.5: Vorinoi Tesselation

The quantization centers used as code book entries can be viewed probabilistically. The distance between a particular piece of data and a quantization center represents a probability. The larger the distance, the less likely that the code book entry accurately describes the data. It's this idea that ties the spatial pooler to Bayesian Networks.

The spatial pooler learns by creating a code book from input data in an unsupervised manner. The goal is to create a set of code book entries that approximately model the input data.

A less than optimal, but simple, way to do this is to define a maximum distance parameter D. If the distance of an input does not fall within D of any of the current code book entries, then a new entry is created using that input data as a new quantization center. Depending on the input data, this method can create code books of indefinite length, so some strategy needs to be used to avoid memory overruns.

After learning is complete, the spatial pooler uses a distance metric to compare incoming data to each code book entry. The entry with the shortest distance is the *winner*. The index of the winner is sent to the temporal pooler for further processing.

3.3.3 Temporal Pooler

The temporal pooler uses time to keep track of relationships between quantization centers sent from the spatial pooler. It groups observed quantization centers into temporally adjacent interconnected sequences.

To do this, the temporal pooler uses a data structure called a Time Adjacency Matrix. This is a $M \times M$ matrix where M is the total number of quantization

centers learned by the spatial pooler.

During operation, the spatial pooler will send the index of the winning code book entry to the temporal pooler. The temporal pooler uses this index and the index from the previous time slice to increment the corresponding entry in the time adjacency matrix.

	a	b	c	d	e
a	0	0	3	1	0
b	1	0	0	0	0
c	0	4	0	0	0
d	0	0	0	0	2
e	0	0	1	1	0

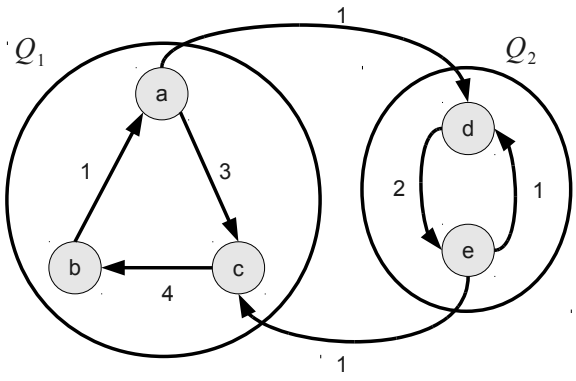

Figure 3.6: Temporal Grouping Example

After training is complete, quantization centers with non-zero entries in the time adjacency matrix are grouped based on their temporal relationships. For example, if a, b, c, d and e are quantization centers, then a possible time adjacency matrix and grouping are shown in Figure 3.6.

The groups Q_1 and Q_2 on the right seem to make sense, but how do we quantify what makes one group better than another group? George [6] shows a measure based on the sum of the connection strengths in each group. Suppose $D = \{D_1, D_2, ...D_N\}$ are disjoint sets of code book entries, and there are n_k elements in partition D_k. Then,

$$t_k = \frac{1}{n_k^2} \sum_{i \in D_k} \sum_{j \in D_k} T_{ij} \tag{3.7}$$

is a measure of the temporal similarity in group k where n_i is the number of quantization centers in that group and T_{ij} is the temporal adjacency matrix. Furthermore,

$$J = \frac{1}{2} \sum_{k=1}^{N_g} n_k t_k \tag{3.8}$$

is the global objective for all groups. Maximizing J produces the best grouping for a set of quantization centers.

The grouping on the right in Figure 3.6 shows the optimal groups according to this metric for the given temporal adjacency matrix. This grouping yields $J \approx 0.819$ which is its maximum value for this set of quantization centers. The goal then is to find the maximum value J for any set of quantization centers.

Given the potentially large numbers of centers, it is not computationally feasible to compute J using brute force methods. Instead, Numenta describes a method based on the idea of the *M top neighbors*.

In this context, if each row in the TA matrix represents a center, then every nonzero column would represent a neighbor, with the largest entries being the top neighbors. This method first chooses a quantization center with the most connections.

In the case of the example in Figure 3.6, a and e have the greatest number of connections, which is 2.

The next step is to find the M neighbors with the largest entries in the transition adjacency matrix. In the example, the two top (and only) neighbors associated

with a are b and d, so these three centers are placed in the first group. Subsequent groups are made by repeating this process until all quantization centers are part of a group.

Note that this group is different from the optimal grouping shown in the example. This highlights the issue with anything less than an exhaustive search for optimal grouping, and that is the possibility of converging on local optima. However, in practice, the full HTM network is robust to these sub-optimal groupings. This will be demonstrated in chapter 6.

After temporal grouping is complete, the temporal pooler matches incoming data to its respective group. The index of the winner is sent to the next level of the hierarchy for further processing. The only exception is the top node where the temporal group represents the final result of the HTM network.

3.4 Hidden Markov Models and HTM

The time adjacency matrix in the temporal pooler is analogous to the transition probability matrix used in Markov Modelling. A Hidden Markov Model is a mathematical tool kit that can be used to model temporal data in the real world [25, 26].

It consists of a set of hidden states $S = \{S_1, S_2, ...S_N\}$. The system moves from state to state during each time slice t. The transition from state to state is probabilistic. The probability of moving from state S_i to state S_j is given by the $N \times N$ state transition matrix A, where each element, a_{ij} is defined as,

$$a_{ij} = P(S_j^{t+1}|S_i^t), \quad 1 \leq i, j \leq N \tag{3.9}$$

and S_i^t means the system is in state i in timestep t. Each state will emit a symbol, $V = \{v_1, v_2, ...v_M\}$ also in a probabilistic manner, creating the set of symbol

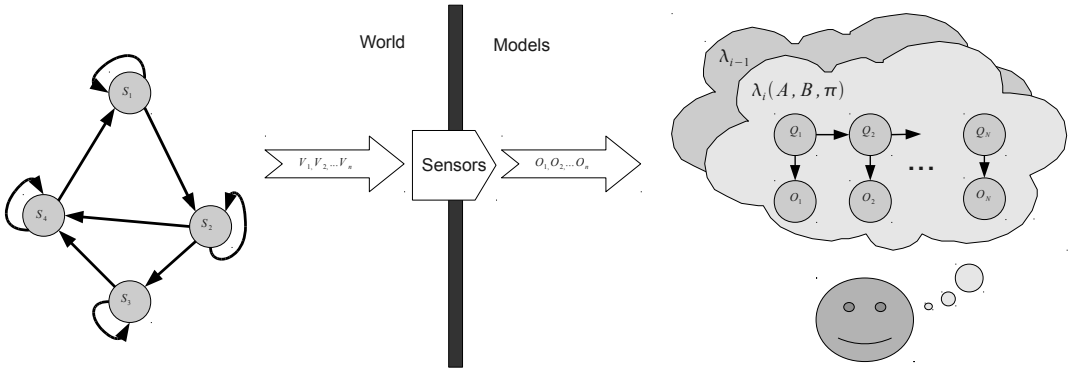

Figure 3.7: Hidden Markov Model

probability distribution functions $B = \{b_1(k), b_2(k), ...b_N(k)\}$, where

$$b_j(k) = P(v_k^t | S_j^t), \quad 1 \leq k \leq M. \tag{3.10}$$

Finally, the initial state distribution is given by $\pi_i = \{\pi_1, \pi_2, ...\pi_N\}$, or equivalently,

$$\pi_i = P(S_i^1). \tag{3.11}$$

Hidden Markov Modeling attempts to use these parts N, M, A, B, π to construct a model $\lambda = (A, B, \pi)$. The objective with HTM is to learn the most probable Markov Model λ, that matches a set of observations $O = \{O_1, O_2, ...O_N\}$,

$$P(O|\lambda) = \sum_{i=1}^{i=N} \alpha_T(i), \tag{3.12}$$

where α_T is defined as the probability some partial observation sequence given the model λ, arriving at observation O_T, during state S_i at time T.

$$\alpha_T(i) = P(O_1, O_2, ...O_T, S_i^T | \lambda) \tag{3.13}$$

During temporal grouping, quantization centers from the spatial pooler are used as observations to create a Markov Chain. The parameters of the model, namely A, B and π are learned in an unsupervised manner from the data using the temporal grouping heuristic described earlier. The HTM algorithm attempts to learn the most probable state that could have produced a given observation.

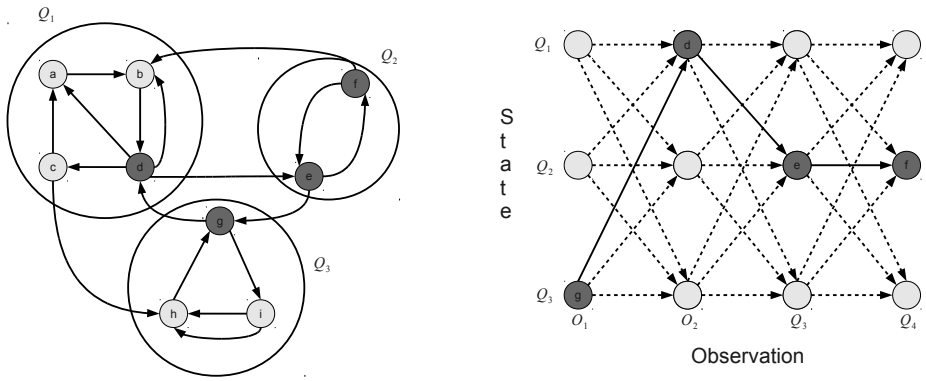

Figure 3.8: Temporal Grouping and Markov Trellis

These groups represent the forward calculation along the Markov Trellis as shown in Figure 3.8.

3.4.1 Training an HTM Network

Since the HTM learning algorithm is dependent on time, static images are insufficient to train the network. Instead, training data must be in the form of *movies*, or sequences of temporally connected images. Training an HTM network involves repeated exposure of these movies to the HTM network.

Each level in the hierarchy gets trained with data from previously trained lower levels. The concept is illustrated in the example network depicted in Figure 3.9. Each node in the hierarchy is shown with its two components, the spatial and

temporal poolers are represented by 'S' and 'T' respectively.

The spatial and temporal poolers have 3 modes, train, test and off. During training mode, the spatial pooler organizes quantization centers based on input data, while the temporal pooler updates the TA matrix and forms temporal groups. During test mode, the spatial pooler sends the index of winning code book entries to the temporal pooler which maps them to a previously learned temporal group. The temporal pooler then sends the index of that group up to the next level of hierarchy. If either pooler is off, no operation is performed and inputs are ignored.

Training an HTM node is done in two phases. During the first phase, the temporal pooler is off, while the spatial pooler is in training mode. The spatial pooler is exposed to the training data, and learns a set of quantization centers.

In the second phase, the spatial pooler is set to run mode, and the temporal pooler is set to training mode. Here, the temporal pooler populates the time adjacency matrix with this data, and performs temporal grouping. Finally, both poolers are placed in run mode, and training of the next hierarchical level can begin.

The example depicted in Figure 3.9 shows an HTM network during the training process. Both the spatial and temporal poolers in level 2 are in run mode, indicated by their dark grey color. In level 1, the spatial pooler is in run mode, and the temporal pooler is in training mode. Level 0 is off, indicated by the color white.

The untrained network starts out by training the lowest level, followed by subsequently higher levels until the top level in the hierarchy is complete. At each iteration, training data is re-presented to the bottom layer.

So far, the training process has been unsupervised, where both spatial and temporal poolers perform some variation of clustering. However, the HTM node

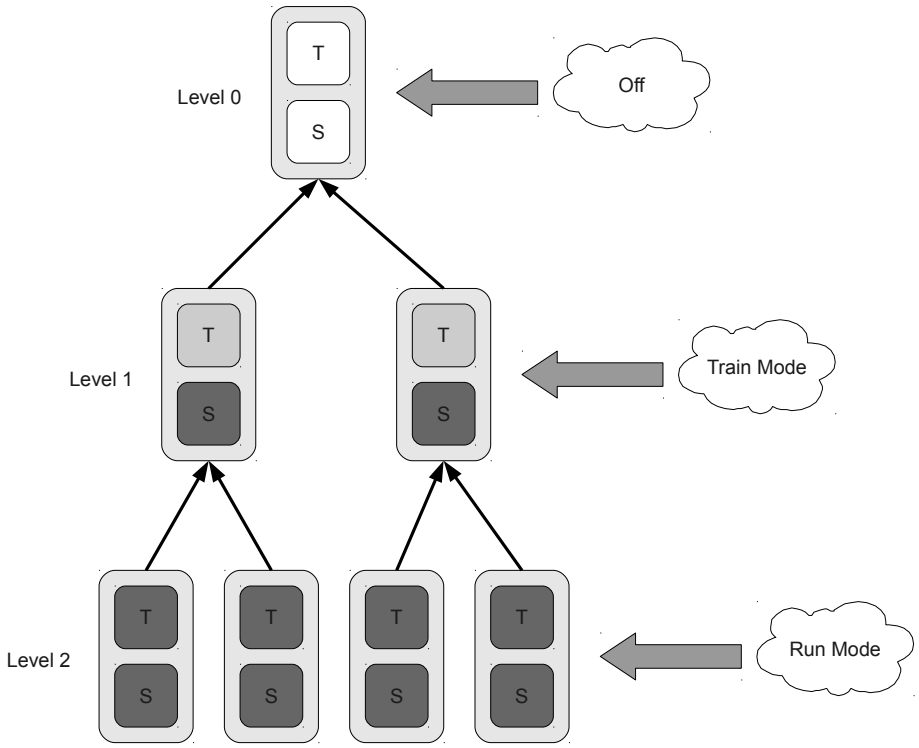

Figure 3.9: Training a 3 level HTM hierarchy

is also capable of supervised learning.

Although semi-supervised methods exist for data clustering that may be appropriate for use in the spatial pooler [8], it is much easier to modify the temporal pooler to perform supervised learning. This is done only in the top layer during the grouping phase of the temporal pooler.

Suppose the training data has been pre-classified. This classification is sent to the top node of the HTM network where the temporal pooler uses it in the temporal grouping process. When the top node performs temporal grouping, all the quantization centers that have non-zero entries in the temporal adjacency matrix are added to the group indicated by the classification information.

This means, if all the training data that are presented to the HTM network are of the same class, then they can all be put into the same temporal group at the top node of the HTM network. Repeating this process for the training of movies of different classifications creates groups in the top node corresponding to the classes that were presented.

3.5 Using K-Means for Spatial Pooling

An improvement to the spatial pooling method discussed above would be to use an algorithm called K-means [2]. Unlike dynamic clustering, the size of the code book is fixed. The K-Means algorithm works on infinite streams of input data, and attempts to find optimal centers in an iterative manner.

The learning process begins with no prior knowledge of the input data other than its dimensionality, and starts by choosing K random points in the input space which become the quantization centers. Incoming feature vectors are then compared to each center, and assigned to the one with the smallest distance which is the winner.

Next, the winner is moved slightly towards the geometric center of its K closest neighbors. This technique can get stuck in local minima depending on the initial position of the centers. K-means is also susceptible to outliers, which can be corrected by using the median instead of the mean, although this requires additional computational complexity.

3.6 HTM and Scale Invariance

As will be shown in the next section, HMAX largely acts like an image processing filter. As such, all of its processing parameters tend to be fixed at run time.

HTM, on the other hand, is a general purpose hierarchical learning algorithm which incorporates both space and time. HTM aligns well with what happens with real world object recognition. As an observer moves closer to an object, the object appears larger, taking up more room in the field of view. If the observer moves laterally with respect to the object, the object moves laterally within the field of view.

HTM works by learning how both of these motions change together in time as the observer moves in relation to observed objects.

Suppose some object is placed before an observer. For example, the left side of Figure 3.10 shows a figurine of a cat, while the right side shows an example of image features that are sent to the HTM network for a small region of the image near the head of the cat. Unlike HMAX, HTM must learn scale invariance.

As the camera moves forward, and successive pictures are taken, two things happen to the object in the field of view. First, the image features associated with the cat figure move slightly from right to left, and bottom to top. A second, more subtle effect is illustrated by the grey arrows, and shows the features associated with the cat taking up larger parts of the field of view in subsequent images. Both these translations are learned by the HTM network.

In the spatial poolers, these changes cause new quantization centers to be written to their respective code books. The temporal poolers then record new transitions in the temporal adjacency matrix.

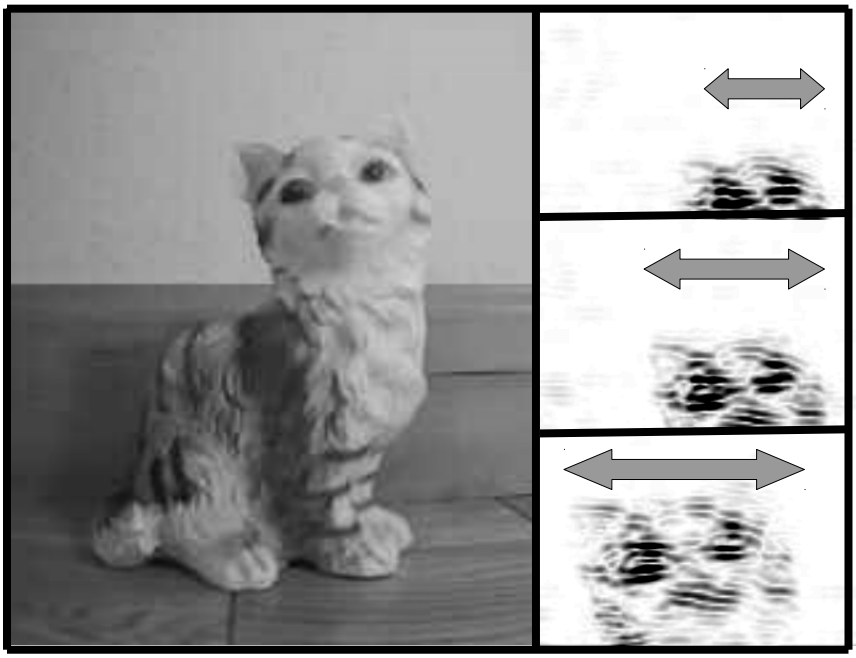

Figure 3.10: Effects of scaling on image features

Chapter 4

The Object Recognition Filter

4.1 Introduction

The visual object recognition task begins with filtering. The image is represented by a 2 dimensional array of pixels, and is operated on by any number of filters, ultimately producing some set of features. These filtering operations generally make heavy use of convolution.

4.2 Convolution and the Pyramid Transform

Image convolution is the basis for many computer vision algorithms. The discrete version is defined as,

$$g(x,y) = H(x,y) * I(x,y) = \sum_{x'=-\infty}^{\infty} \sum_{y'=-\infty}^{\infty} H(x-x', y-y')I(x,y) \tag{4.1}$$

where I(x,y) is the input image and H(x,y) is another usually smaller image called a *convolution kernel*. The process creates a third image $g(x,y)$ as shown in figure 4.1.

Generally, the convolution of two images, $M_1 \times N_1$ and $M_2 \times N_2$ yields a third image $(M_1 + M_2 - 1) \times (N_1 + N_2 - 1)$. This means that in practical systems, pixel corruption can occur around the borders pixels.

Pixel Corruption can be alleviated through a number of common techniques including wrapping, or by setting calculations that fall outside the overlapped

image regions to zero.

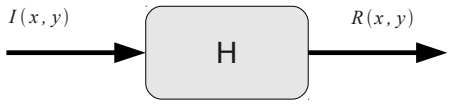

Figure 4.1: Convolution

The Gaussian kernel is the basis for many computer vision applications. The operation is called a *Gaussian blur,* and is used for low pass filtering or smoothing operations. It is used as an interpolation step for sub-sampling or reducing the size of an image. The Gaussian kernel is defined as,

$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{4.2}$$

where σ is the standard deviation of the Gaussian distribution.

Burt and Adelson [3, 1] pioneered the image pyramid, an early application of hierarchical Gaussian image filtering. First, the image is sub sampled, or *reduced*, and then up-sampled, or *expanded*. The expanded image is then interpolated by convolving it with a Gaussian kernel. The process is repeated to the desired depth.

This can be thought of as an image transform where each successive value for σ describes Gaussian "weighting" functions $w_l(m, n) = G(m, n, \sigma_l)$ which are transform code elements. Reduce operation smooths and reduces the image.

$$P_{l+1}(x, y) = \sum_m \sum_n w_l(m, n) P_l(2x + m, 2y + n) \tag{4.3}$$

Conversely, the expand operation expands a previously reduced level.

$$P_l(x,y) = 4\sum_m \sum_n w_{l+1}(m,n) P_{l+1}(\frac{x+m}{2}, \frac{y+n}{2}) \tag{4.4}$$

In both cases $P_0(m,n)$ represents the original image.

Witkins was the first to coin the term "scale space" [36].

$$L(x,y,\sigma) = G(x,y,\sigma) * I(x,y) \tag{4.5}$$

Scale space representations can be derived as the solution to the linear diffusion equation.

$$\frac{\partial G}{\partial \sigma} = \sigma \nabla^2 G = \sigma \Delta G = \sigma(\frac{\partial^2 G}{\partial x^2}, \frac{\partial^2 G}{\partial y^2}) \tag{4.6}$$

Subtracting successive sets of neighboring scales provides a difference approximation to $\partial G/\partial \sigma$. Furthermore, since the Gaussian kernel is separable, the one dimensional result carries over into higher dimensions. That is, the one dimensional solution is applied to successive rows or columns of a two dimensional image.

In addition to their usefulness in traditional image processing, Gaussian functions have a strong connection to Neurobiology. Experimental data suggests neural interaction in the brain can be modeled using Gaussian functions. For example the distribution of inhibitory neurons in the primary visual cortex is modeled using a difference of Gaussian (DOG) function [32].

A related image pyramid technique is the Laplacian Pyramid. Using this method, the image is first convolved with a Gaussian kernel creating a new image. Next the original image is subtracted from the new image creating the difference of Gaussian. The Gaussian kernel can be used to detect edges, ridges and corners.

The second and third order directional derivatives are computed:

$$D(x, y) = [G(x, y, k\sigma) - G(x, y, \sigma)] * I(x, y) \tag{4.7}$$

Where k is a multiplicative factor $k < 1$.

Like the Fourier transform, the Laplacian pyramid is sensitive to different scales, and thus, different spatial frequencies. Unlike the Fourier transform, however, code elements remain spatially fixed. This makes the transform useful for picking out image features that are scale invariant.

Furthermore, by carefully quantizing the pixels in the image pyramid, a reduced entropy version of the image is constructed. This way, the image pyramid performs feature enhancement, and provides a compact representation.

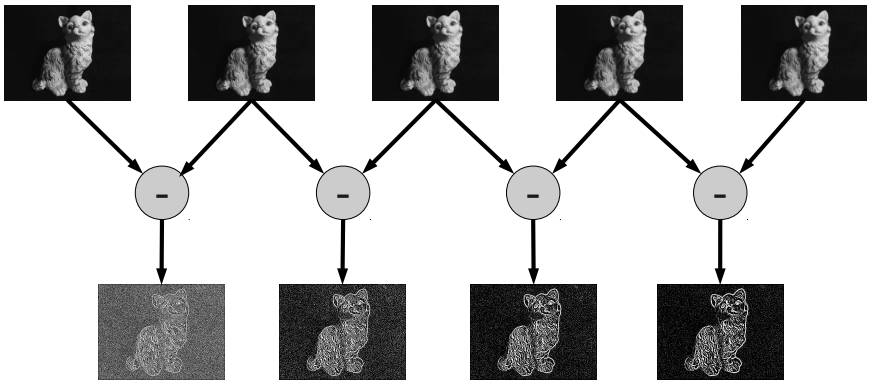

Figure 4.2: Image Pyramid

4.3 Scale Invariant Features

The image pyramid can be used to extract scale invariant features used to represent objects in an image. This technique was pioneered by David Lowe, [17] who created the Scale Invariant Feature Transform (SIFT). SIFT creates a series of DOG filters which employ Gaussian kernels of successively increasing width. This creates a

scale-space representation of object features that correspond to the scales of the image pyramid.

In scale space, each pixel has a set of $K = \{k_0, k_1, ..., k_L\}$ pixels that represent the results of DOG filtering for each scale. A set of sample points or *keys* are specified over the image. A feature is created by finding the maximum value in a 3 X 3 region centered about a key point, and extended through scale space. The algorithm then searches for local extrema within all the samples in the image. Since local extrema can be arbitrarily close together, there's a trade-off between sample frequency and detection rate.

Keypoints have an associated image gradient vector, the magnitude and orientation of which is chosen as a peak in a histogram of neighboring values using a Gaussian distributed set of weights. The method has been shown to create keypoint attributes which are robust to affine image translations such as rotation and scale. Keypoints in a new image then can be compared with previously stored keypoints by vector subtraction. Again SIFT uses a peak histogram method using pixel values.

Neighboring keypoints are determined using a variation on the kd-tree algorithm called best-bin-first which is designed to efficiently find approximate solutions in high dimensional search spaces in a probabilistic manner.

At this point, a set of hypotheses is formulated using a generalized Hough transform. The hypotheses consist of groups of keys that agree on a particular geometric pose. Next, a hash table is used to match the hypothesis with model rotation orientation and scale. Prospective key point locations are then compared to model locations using a least squares method, and outlying keys are discarded.

4.4 HMAX, Revisiting the Visual Cortex

Just as Lowe introduced SIFT in 1999, Riesenhuber and Poggio [27] were introducing another model of object recognition based on the work of Hubel and Wiesel on the neurobiology of the visual cortex. The model, dubbed HMAX, is inspired by the interactions between simple and complex cells in the visual cortex.

HMAX creates invariant features by pooling information between hierarchical levels. In a pooling operation, complex cells in higher hierarchical levels receive input from a much larger group of simple cells in lower hierarchical levels defined by that cell's receptive field. This model is said to be *feed forward* because information flows one way from the point where data enters the system, to the final layer in the hierarchy.

HMAX uses two kinds of computations, a linear summation (SUM), and the maximum (MAX), with the idea of building complex invariant feature detectors from hierarchies of simpler localized modular units. Both SUM and MAX are modeled based on current understanding of the behavior of neurons in the visual cortex. However, the major claim of this model and supporting research is that the maximum (MAX) operation is the primary mechanism for object recognition in the cortex. The SUM operation is a linear sum with equal weights, and the MAX operation performs a pooling operation similar to finding the maximum of inputs or *afferents*.

4.4.1 S1 Level

The simplest model uses 4 levels of hierarchy in alternating simple and complex layers. The first level, S1, contains filters which employ the first derivative of a Gaussian at some fixed number of rotations. Convolution with these filters

produces an enhanced response for image features that have the same rotation as the filter. In addition, a scale space representation of the input is created by varying the standard deviation of the S1 filters. All together, the local receptive fields of the C1 units cover the input image.

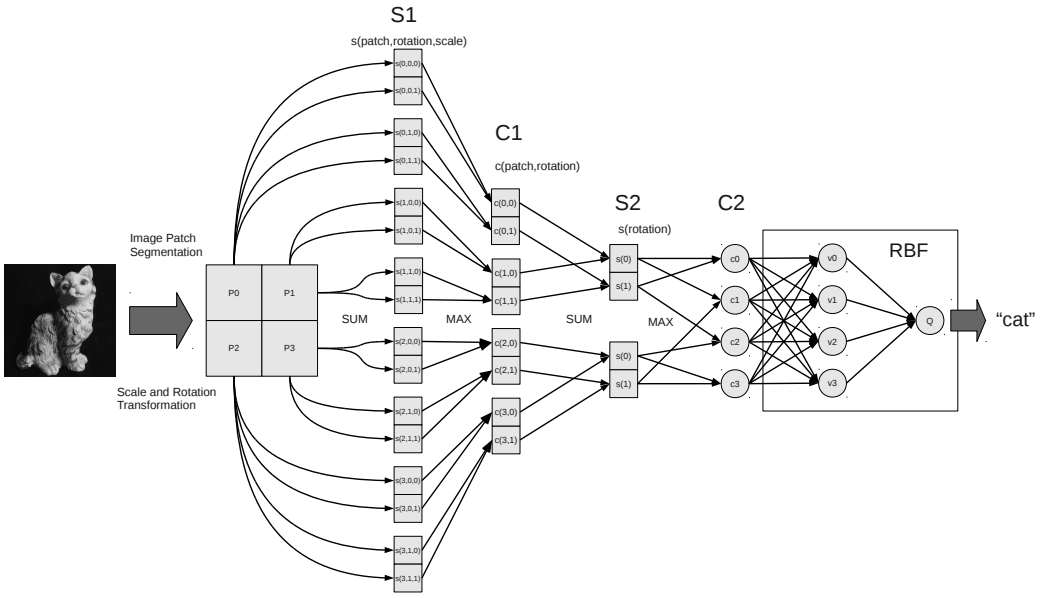

Figure 4.3: Simplified Riesenhuber and Poggio Model with two scales and two rotations.

4.4.2 C1 Level

In the next level, pooling occurs over a larger receptive field encompassing regional inputs from S1 and covering all scales. Pooling in C1 occurs in a local image region and creates features that are scale translation invariant over that region. The output of a C1 unit is the maximum over both scale and feature orientation transforms created in S1.

$$s2 = max \begin{pmatrix} s1_{0,0} & s1_{0,1} & \cdots & s1_{0,N-1} \\ s1_{1,0} & s1_{1,1} & \cdots & s1_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ s1_{M-1,0} & s1_{M-1,1} & \cdots & s1_{M-1,N-1} \end{pmatrix}$$

where each element, $s1_{m,n}$ is an $N \times N$ image section. This is the second MAX operation shown in figure 4.3.

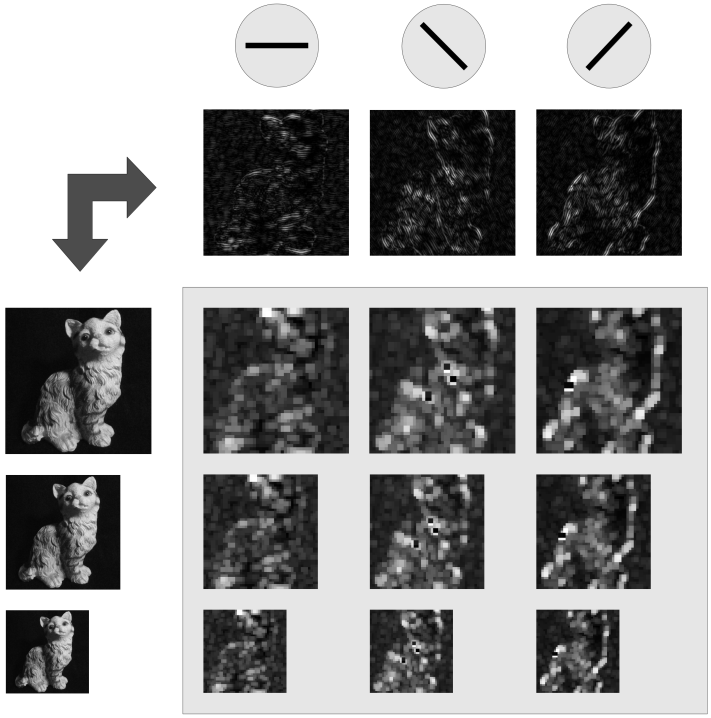

Figure 4.4: Feature Orientation -vs- Scale

4.4.3 S2 Level

The S2 layer combines features from the lower levels to create *composite features*. Early versions of the algorithm combined two types of features from C1, but this

was found to be susceptible to problems associated with image clutter and noise. The revised version combines all feature types from a group of neighboring C1 units. The response of an S2 unit is defined as

$$s2 = exp\left(\frac{-\sum_{k=0}^{N-1}(c1_k - 1)^2}{N}\right) \tag{4.8}$$

Where N is the total number of composite features.

4.4.4 C2 Level

The final layer has a unit for each combination of edge orientation in the neighborhood of C1 units. For example, the C1 neighborhood consisted of a 2x2 square and there were 4 rotations, then there would be 4^4 or 256 units in S2.

4.5 Serre's Contribution

Later, Serre developed a model derived from Poggio's earlier work that added a number of extensions, while improving both biological plausibility and classification performance [30]. The first layer uses Gabor filters instead of Gaussian derivatives. The S2 layer uses the *bag of words* [4] concept.

Serre's model, like earlier version of HMAX is hierarchical and uses 4 processing layers. In this model however, each subsequent layer has fewer units and sub sampling is used to do this.

4.5.1 S1 Improvements

At the S1 level Gabor filters are used instead of Gaussian Derivatives because they have a number of advantages over Gaussian derivatives [29]. Interestingly, the parametric equations

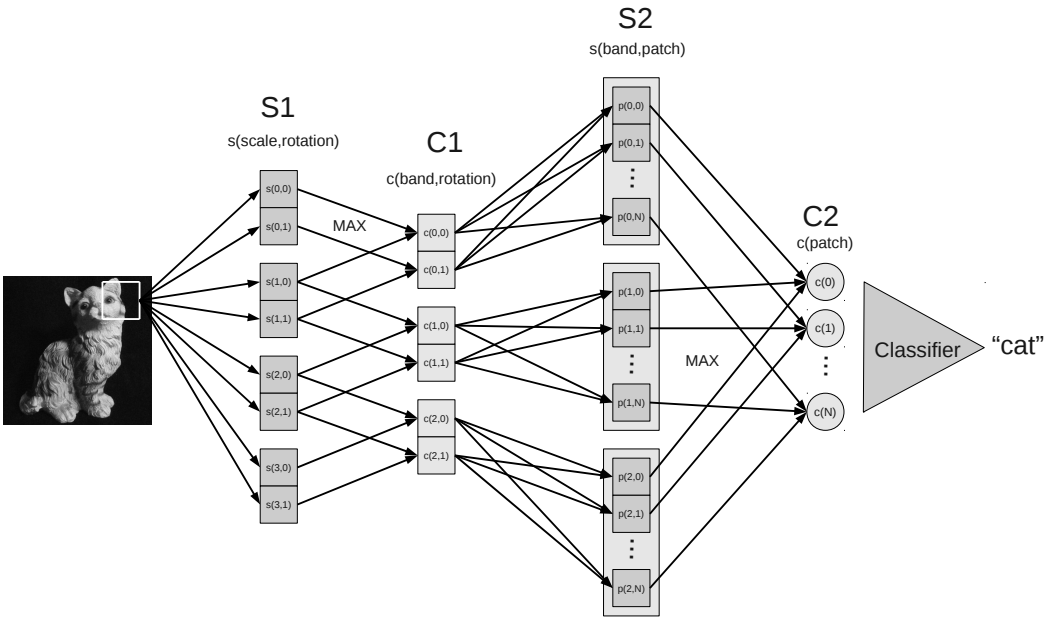

Figure 4.5: Simplified Serre model with two rotations and two scales

$$\begin{aligned} X &= x\cos(\theta) - y\sin(\theta) \\ Y &= x\sin(\theta) + y\cos(\theta) \end{aligned} \tag{4.9}$$

Can be used to describe both the second derivative of the Gaussian function

$$G_d(x,y) = exp\left(\frac{-X^2+\gamma Y^2}{2\sigma^2}\right)\frac{X}{\sigma^2(\sigma^2-1)} \tag{4.10}$$

as well as the Gabor function.

$$G_g(x,y) = exp\left(\frac{-X^2+\gamma Y^2}{2\sigma^2}\right)cos\left(\frac{2\pi}{\gamma}X\right) \tag{4.11}$$

Using Gabor functions in S1 provides more specificity for orientation and frequency.

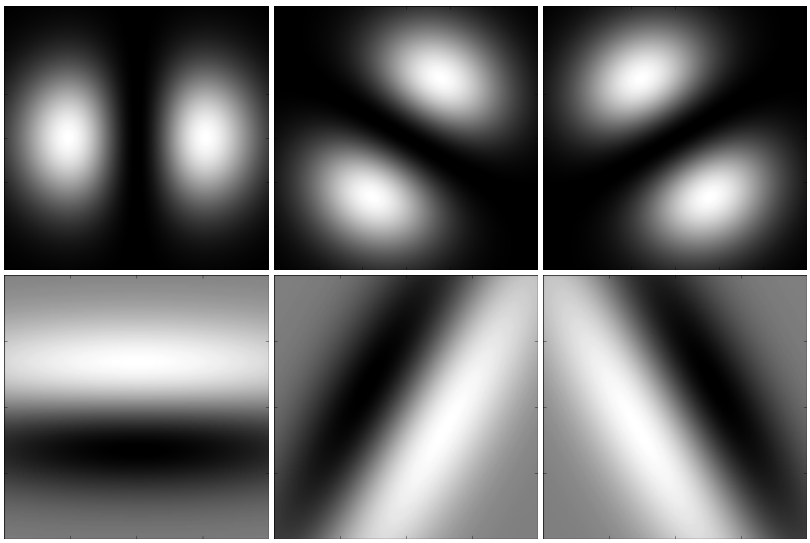

Figure 4.6: First derivative of Gaussian (top), Gabor (bottom)

4.5.2 C1 Level

Inputs from S1 are sub-sampled and a the $y = max(x)$ function is applied over neighbors. In this model sub-sampling has the effect of reducing the number of units in higher hierarchical layers. After pooling neighboring inputs, pooling is again performed between subsequent scales creating *bands* in C1.

$$c1 = \max(s1_a, s1_b) \tag{4.12}$$

where $s1_a$ and $s1_b$ are two adjacent scales in layer S1.

4.5.3 S2 Level Improvements

The biggest change from the original model is in the S2 level. Pattern matching is still used, but instead of simply building composite features, the new model uses a set of N *prototypes* that are compared to each input from the C1 level. In practice N can be on the order of 1000. For each band, units in level S2 perform an RBF like operation over the orientation of inputs from C1 that fall within a particular unit's receptive field.

$$s2 = \exp\left(\beta \|c1 - P_i\|^2\right) \tag{4.13}$$

The result are $N \times M$ units in S2 where N is the number of prototypes and M is the number of scales. The prototypes are generated by randomly selecting C1 units after presenting an image to the model. Many such prototypes are collected by using many images taken from random natural scenes creating a *visual vocabulary of features*.

4.5.4 C2 Level

The final level performs the MAX() operation over S2 inputs and over each scale. This results in N outputs, one for each patch. This constitutes the output of the HMAX model. At this point, the authors used a Support Vector Machine to do the final classification, however in theory, any learning algorithm could be used.

4.6 Intelligent Signal Processing

The techniques discussed in this chapter are all examples of Intelligent Signal Processing or ISP. ISP uses adaptive techniques such as learning and complex

models of data distributions to extract and represent maximum information from a signal [9]. These *intelligent filters* can be used as a pre-processing step for higher level models. The remaining chapters discus how this is done using the two classes of cortex inspired models discussed so far.

Chapter 5

Experiments

5.1 Cortex Inspired Object Recognition

The two paradigms discussed so far are both Cortex inspired, and based on current understanding the visual cortex. However, brain inspired models like HTM and HMAX tackle the overall object recognition problem from different directions. Image pyramid based methods like HMAX and SIFT make explicit use of scale, but have no sense of time. HTM, on the other hand, only indirectly tackles scale, but makes explicit use of time. In this chapter, I combine the two models to make an object recognition model that explicitly uses both scale and time to recognize objects in temporal sequences of images using models inspired by the Cortex.

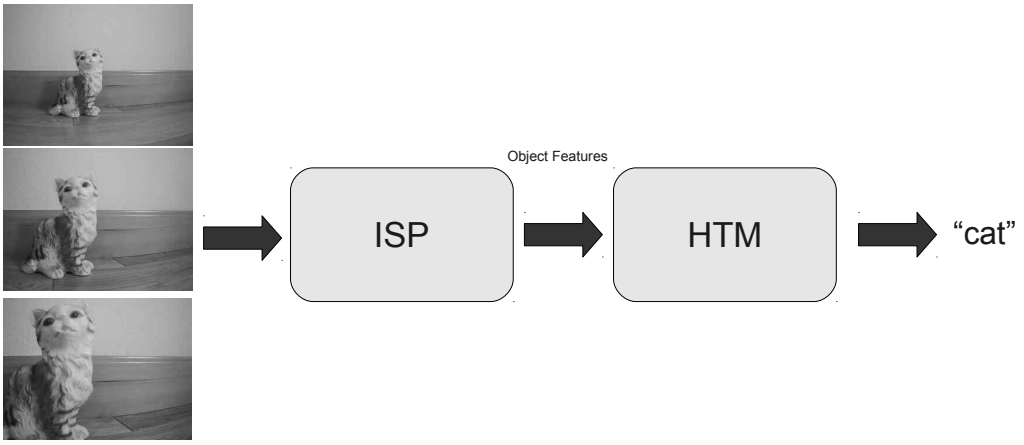

Figure 5.1: Bio-Inspired Object Recognition

Since HMAX and its predecessors were designed to model the *early visual processing* of Cortex, it should be near the raw visual data. This constitutes an intelligent signal processing block. The HTM model receives input from the ISP

filter in the form of image features. Since HTM can perform both supervised and unsupervised learning, it is used to do the final object classification.

5.2 The Experiment Software

To test these ideas, I constructed software for use in a robotics platform. The system uses a WiFi enabled robot to communicate to computer vision software residing on a standard PC. The software performs object recognition while receiving successive images from the robot via the WiFi network.

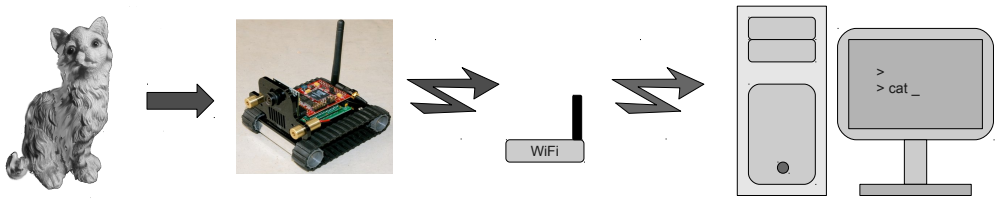

Figure 5.2: Robot Vision System

The software, written in C++, consists of three modules, including the robot interface along with the HMAX and HTM implementations [1].

5.2.1 The SRV1 Robot

This inexpensive, open-source robot comes with 802.11B WIFI and a camera. It moves via two independently controlled tracks. It has a 500 Mhz Analog Devices Blackfin processor on board. Commands are issued over the TCP/IP interface. The robot is capable of a wide range of tasks such as movement control and image capture, analysis and transfer. For these experiments, images are transferred from

[1]For a detailed description of software components see Appendix 1

the robot for analysis, while movement commands are sent to control the robot's position.

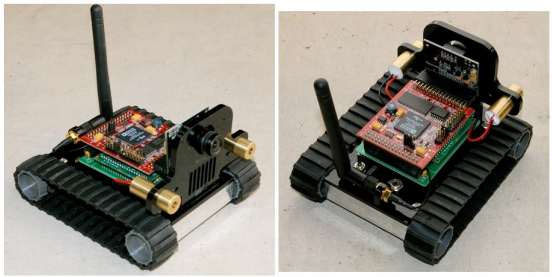

Figure 5.3: SRV1 Open Source Robot

The robotic interface has methods for grabbing and decoding video as well as controlling the robot's movement using TCP/IP protocol. I shared this library using Google Code with the SRV1 community. The robot interface is instantiated as a link object by the main routine. The main routine uses OpenCV to read video from the robot or from the local disk and can switch from video mode to using still images. OpenCV allows tight integration of the various software components by providing a common image format.

5.2.2 HMAX Filter

For the purposes of this investigation, I developed a piece of software based on HMAX. This is a single threaded C++ implementation. The original HMAX source code makes extensive use of Matlab for various administrative tasks such as creating the image pyramid and calculating receptive field sizes. I created a version of this wrapper code using C++ and OpenCV that implements all the original functionality, plus extra functionality like being able to dump an image from between layers.

The wrapper for layer S2 was the most challenging part. This layer uses a

rather large feature library that must be loaded at run time. The original code used Matlab to do this work and used a Matlab binary file format for the library itself. My C++ implementation uses a plain text format for the library that is easier to understand and modify.

I created a version of the completed HMAX/OpenCV mash-up and shared with others in the HMAX community via a forum page hosted by Jim Mutch at MIT. In the forum, I met another researcher who works visual object recognition. Together, we started a Google code project that makes available two pure C++ implementations of HMAX.

5.3 The Experiments

The experiments investigate three vision models using a scale invariant object recognition task. The models are shown in Figure 5.5 and consist of an HMAX layer along with a filter layer.

In each of the three experiments, the filter layer is constructed using modular HMAX sub-layers as components. The final experiment, experiment 3, uses the full HMAX implementation. In all three experiments, 6 rotations are used in the S1 sub-layer. The filter layer output is a set of feature vectors with real components ranging from 0.0 to 1.0.

For the three experiments, the same HTM network is used. Data from the filter layer feeds into the base of the HTM network. The vector elements from the filter layer are first quantized into 255 levels before being sent to the HTM network. In each experiment, the filter layer output is also up-sampled using linear interpolation to fit the requirements of the HTM network base layer geometry.

The HTM network is set up in a hierarchy, with nodes in level N taking input

from 4 child nodes in level $N-1$. The network has 5 levels. A single node at the top of the hierarchy is responsible for making the final classification.

The best results were obtained using four different figurines. The robot was placed 1.5 Meters from the figurine. A series of commands were issued to the robot to drive forward for approximately half a second, stop and take a picture which was saved to a computer file. In each successive picture, the robot moves closer to the figurine. This process was repeated for each of the four figurines.

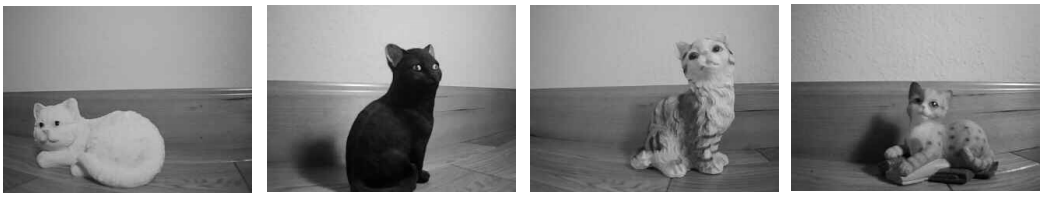

Figure 5.4: Test Subjects

Three object recognition experiments were performed using input from the SRV1 robot. The objects were plastic cat shaped figurines purchased at the Dollar Store. The experiment data consist of image sequences taken as the robot approached the object of interest. There are 20 frames in each sequence or *movie*. Four separate movie sequences were made for each figurine. Three movies were used for training and one for classification.

In each experiment, the HTM network receives a scaled output from the filter layer. Each HTM node on the bottom level takes input from a non-overlapping region of the input image. This window is used to create the input vector for a particular node in the base layer of the HTM hierarchy.

The three experiments are outlined in table 5.1 and figure **??**.

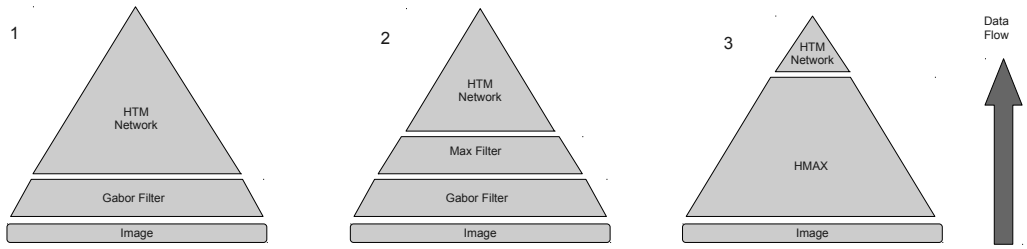

Figure 5.5: The three experiments; illustrating increasingly sophisticated signal processing blocks.

	HMAX Output	Node Window	Input Vector
Experiment 1	320x240	15x20	1800
Experiment 2	64x48	3x4	72
Experiment 3	91x91	6x6	36

Table 5.1: Experiments

5.3.1 Experiment 1

Experiment 1 uses only the S1 filter layer which sends a 320x240 array of features to the HTM layer. Each node in the lowest layer of the HTM network processes a non-overlapping 15x20 window on this data. Since 6 rotations are used this creates a feature vector of length 1800.

5.3.2 Experiment 2

Experiment 2 uses a filter with two layers, S1 and S2. The output is a 64x48 array of features which is sent to the HTM layer. Each node in the lowest layer of the HTM network processes a non-overlapping 3x4 window on this data. Again, 6 rotations are used which creates a feature vector of length 72.

	S1	C1	S2	C2
1	320x240	310x230	61x45	45x58
2	268x200	258x190	49x37	37x46
3	226x168	216x158	41x29	29x38
4	190x142	180x132	35x25	25x32
5	160x120	150x110	29x21	21x26
6	134x100	124x90	23x17	17x20
7	112x84	102x74	19x13	13x16
8	94x70	84x60	15x11	11x12
9	80x60	70x50	13x9	9x10
10	66x50	56x40	9x7	7x6
11	56x42	46x32	7x5	5x4
12	46x34	36x24	5x3	3x2

Table 5.2: Image Scales used in full HMAX implementation

5.3.3 Experiment 3

Experiment 3 uses all 4 layers of HMAX, namely S1, C1, S2, and C2. Note that Experiments 1 and 2 only make use of one 1 image pyramid level. Experiment 3 uses a complete HMAX implementation including a 12 layer image pyramid like the one in figure 5.2.

The output is a 1x8150 array of features which is sent to the HTM layer. Also, since layer C2 computes output based on the model's response to the feature dictionary, all the image's original topological information is lost. Finally, since HMAX produces a large 1D vector as output, experiment 3 converts this to an appropriate sized array for further processing by HTM.

Chapter 6

Results and Conclusions

6.1 Results

Once the movies were collected, the three experiments were run on a standard PC. Although it would have been possible to run the robot in real time, the models were not implemented with real time performance in mind. Experiment 3 in particular took around 20 seconds to process one image frame. Obviously this process could be improved using pluralization, but the purpose of these experiments remain to show that robot based object recognition is possible with these models.

Considerable time was spent adjusting the spatial pooler threshold for optimal performance. If the threshold is set too large, then not enough code book entries are created to accurately represent the input data. If the threshold is set too small, code books get too big and the system looses the ability to generalize. Finding the right balance took some trial and error.

6.1.1 HTM level 1, Spatial Pooler's view

The lowest level of the HTM pyramid is the first level to be exposed to incoming data from the filter layer. The vector quantizing calculations of the spatial pooler, reduce the incoming features to a 16x16 array of integer data. This provides a compact and convenient method for analysis of large amounts of feature data created by the filter layer.

The Fully trained spatial pooler responds to an input vector by finding the closest code book entry. This is shown for the level 1 spatial pooler in figure 6.1

which illustrates the self-organizing capability of the spatial pooler. The shade of gray represents the distance from the input vector to the closest entry. Dark shades represent small distances while lighter shades represent larger distances. Several sequential input frames are shown with the distance plots generated for all three experiments.

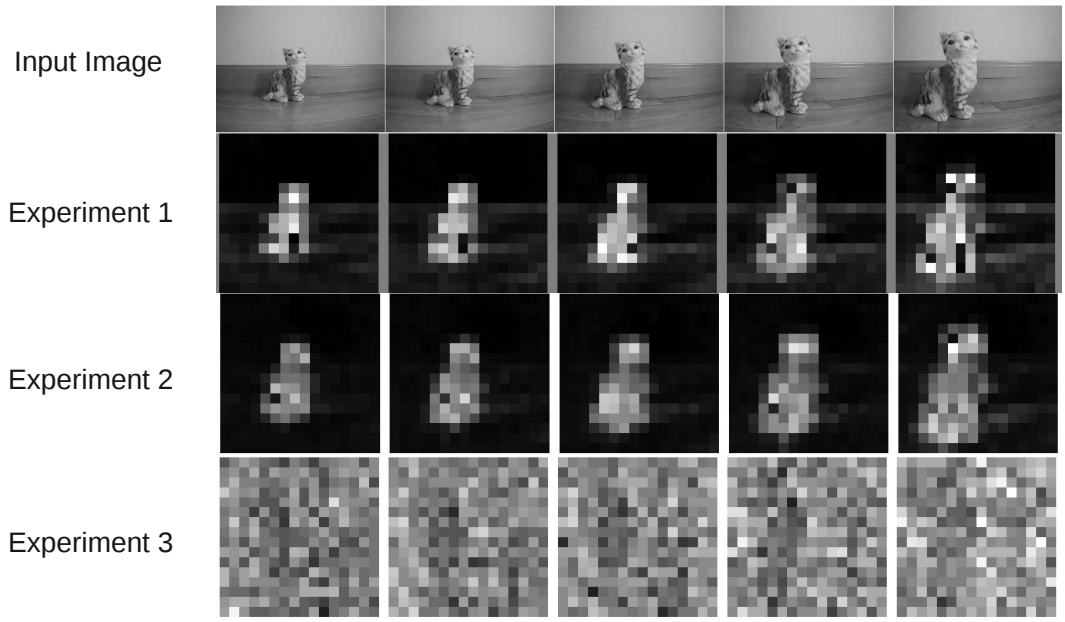

Figure 6.1: HTM Level 1, spatial pooler output

The three experiments show increasingly invariant representations. The image for Experiment 3 showing very little change as the robot moved forward in time. This can be measured by calculating the variance of the data being received by the HTM layer. Table 6.1 shows the average variance for all three experiments and all four objects (cats).

	Cat 1	Cat 2	Cat 3	Cat 4
Experiment 1	1969.178	2195.728	2841.636	2380.302
Experiment 2	138.899	129.912	168.837	133.241
Experiment 3	18.055	11.085	12.021	79.965

Table 6.1: Mean Variance, HTM level 1

	1	2	3	4
Cat 1	0	23	25	0
Cat 2	0	26	22	0
Cat 3	0	19	29	0
Cat 4	0	18	25	5

Table 6.2: Experiment 1, Confusion Matrix

6.1.2 Object Recognition

The ultimate task of this system is scale invariant object recognition through time. The three experiments use 3 movies for training, and 3 movies for test. Each movie consists of 20 sequential images taken by the robot as it moved closer to one of the four figurines.

Three standard metrics are used to compare object recognition performance, accuracy, recall and precision. Accuracy is proportion of classifications, over all examples, that were correct. Recall is the proportion of positive examples that were classified correctly. Precision is the proportion of correct positive classifications over all positive classifications.

Experiment 1

The confusion matrix for Experiment 1, shown in Table 6.2, shows no true positive results for cat 1 and only a few for cat 4.

Dispite this, Table 6.3 shows the accuracy for cat 1 and 4 remains relitively

	1	2	3	4
Accuracy	0.75	0.57291667	0.52604167	0.77604167
Recall	0	0.54166667	0.60416667	0.10416667
Precision	0	0.30232558	0.28712871	1

Table 6.3: Experiment 1, Object Recognition Results

	1	2	3	4
Cat 1	0	9	38	1
Cat 2	0	2	37	9
Cat 3	0	4	37	7
Cat 4	0	3	15	30

Table 6.4: Experiment 2, Confusion Matrix

high. This is due to the fact the model retained a fair ability to correctly classify negitive examples.

Experiment 2

The confusion matrix for Experiment 2, shown in table 6.4, again shows no true positive results for cat 1.

The overall results for Experiment 2, shown in table 6.5, show a slight improvement over Experiment 1.

	1	2	3	4
Accuracy	0.75	0.67708333	0.47395833	0.81770833
Recall	0	0.04166667	0.77083333	0.625
Precision	0	0.11111111	0.29133858	0.63829787

Table 6.5: Experiment 2, Object Recognition Results

	1	2	3	4
Cat 1	47	1	0	0
Cat 2	0	27	20	1
Cat 3	0	1	47	0
Cat 4	0	0	19	29

Table 6.6: Experiment 3, Confusion Matrix

	1	2	3	4
Accuracy	0.99479167	0.88020833	0.79166667	0.89583333
Recall	0.97916667	0.5625	0.97916667	0.60416667
Precision	1	0.93103448	0.54651163	0.96666667

Table 6.7: Experiment 3, Object Recognition Results

Experiment 3

The confusion matrix for Experiment 3, shown in table 6.6, shows a good ability to recognize all 4 cats.

The overall results for Experiment 3, shown in table 6.7, show good accuracy recall and precision.

6.1.3 Analysis

Why did Experiment 3 do so much better than the other 2? Experiments 1 and 2 did not use an image pyramid, and thus showed very little scale invariance. Also, experiments 1 and 2 also don't make use of HMAX level 2 (S2 and C2), which creates complex invariant features. Without these complex features, HTM has to work that much harder.

Despite the results from Experiment 1 and 2, I believe that given a larger memory model, they would have performed better. This is because experiments 1 and 2 used significantly more memory than Experiment 3 as shown in Table 6.8.

	level 0	level 1	level 2	level 3	level 4	Total
Experiment 1	368	1260	4804	23476	68236200	68266108
Experiment 2	484	1636	7220	30888	3083472	3123700
Experiment 3	424	1308	4024	10604	189576	205936

Table 6.8: HTM Memory Usage by Hierarchical Layer

Another clue to the performance of the system in Experiment 3 comes from the Temporal Pooler. In experiments 1 and 2, there was a great deal of variance from one feature frame to the next. This caused nodes, especially in the lower hierarchies, to produce temporal adjacency matrices that are almost completely diagonal. When this happens, the temporal grouping algorithm does not work very well, since it is trying to link various temporal patterns that share rows or columns in the TA matrix. If the matrix is diagonal, then a new temporal group is created for each input from the Spatial Pooler, and thus, no mulit-member groups at all are created.

6.2 Conclusions

I have shown that biologically inspired computational models can be used, and even combined to recognize objects in a robotic environment. The experiments demonstrate that the algorithms are effective when working with real-world data.

In order to work with real world visual data, a feature extraction step is required. Compared to other feature extractors like SIFT, HMAX is a remarkably simple and elegant design. The layers are based on rather simple filtering components that would be familiar to anyone with an understanding of modern image processing.

One of the most significant findings of this experiment is the importance of

the image pyramid. Because of the image pyramid's ability to both create feature rich representations, combined with entropy reduction capability and a convenient model for calculating scale and distance, as well as biological plausibility, it could be considered the "Swiss Army Knife of Image Processing".

Another important result is the link between feature invariance and HTM performance. High feature variance has several negative impacts on HTM including exponentially increasing memory requirements of the Temporal Pooler, and problems with lots of diagonal entries in the TA matrix. Future versions of this algorithm would have to ensure that image features remained as invariant as possible.

Finally, the performance of the object recognition tasks in all three experiments confirmed my hypothesis that HMAX and HTM can be combined to create a scale invariant object recognition system. Furthermore, this system provides a model for scale invariance as it pertains to vision and the brain.

6.3 Future Directions

I plan to use these results as a basis for further research into biologically inspired image processing and visual robotic navigation. Future versions of this work will combine the two models (HTM and HMAX) into a more cohesive, self-organizing system. One step is to incorporate feedback into the model where higher layers in the hierarchy are in a closed loop with lower layers and information travels both up and down the hierarchy.

I also intend to implement at least part of this model as a spiking neural network. There are three reasons for this. First, spiking models have been shown to use less power than conventional computing models. Secondly, spiking models may provide a convenient way to incorporate time into the computational fabric.

Finally, spiking models are a good starting point for uncovering the abilities of non-traditional computing and applying these to real world data.

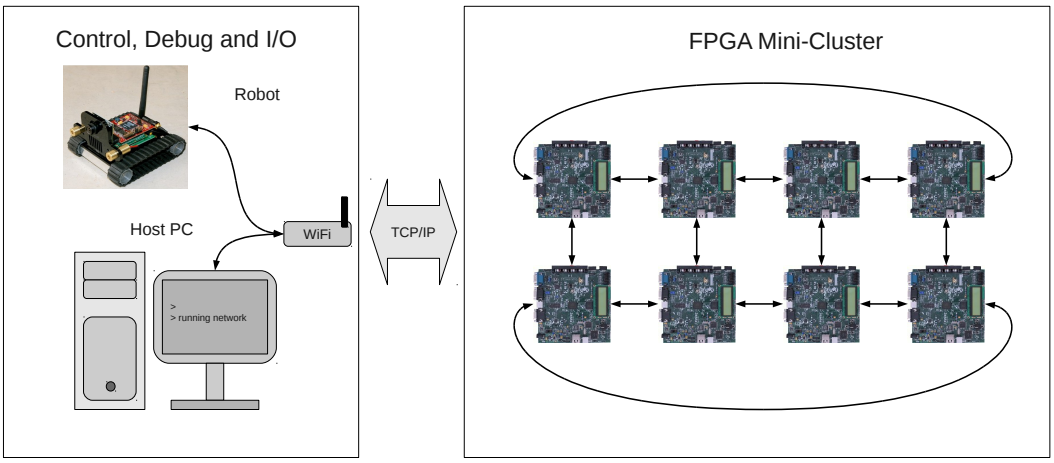

Figure 6.2: Biologically Inspired Computing System

One possible system, inspired by my work with DARPA SyNAPSE, is shown in Figure 6.2. Here, the robot sends image data to a computer which contains NVIDA CUDA. A CUDA-enabled version of HMAX will do feature extraction in real time. This data will then be sent to a spiking neural network "brain" which will implement a hierarchical, self organizing classifier similar to HTM which will employ associative memory to perform both spatial and temporal pooling. Results are then fed back to the PC where appropriate commands are sent to the robot. The system will be used to perform robotic navigation experiments using biologically inspired computing models.

References

[1] E.H. Adelson, C.H. Anderson, J.R. Bergen, P.J. Burt, and J.M. Ogden. Pyramid methods in image processing. *RCA engineer*, 29(6):33–41, 1984.

[2] K. Alsabti, S. Ranka, and V. Singh. An efficient k-means clustering algorithm. 1997.

[3] P. Burt and E. Adelson. The Laplacian pyramid as a compact image code. *Communications, IEEE Transactions on*, 31(4):532–540, 1983.

[4] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, volume 1, page 22, 2004.

[5] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. 2003.

[6] D. George and J. Hawkins. A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In *Neural Networks, 2005. IJCNN'05. Proceedings. 2005 IEEE International Joint Conference on*, volume 3, pages 1812–1817. Ieee, 2005.

[7] P. Gibbons, P. Culverhouse, and G. Bugmann. Visual identification of grasp locations on clothing for a personal robot. *Towards Autonomous Robotic Systems (TAROS*, pages 78–81, 2009.

[8] N. Grira, M. Crucianu, and N. Boujemaa. Unsupervised and semi-supervised clustering: a brief survey. *A Review of Machine Learning Techniques for*

Processing Multimedia Content, Report of the MUSCLE European Network of Excellence (FP6), 2004.

[9] D. Hammerstrom. Computational neurobiology meets semiconductor engineering. In *Multiple-Valued Logic, 2000.(ISMVL 2000) Proceedings. 30th IEEE International Symposium on*, pages 3–12. IEEE, 2000.

[10] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[11] D.A. Hinkle and C.E. Connor. Three-dimensional orientation tuning in macaque area V4. *nature neuroscience*, 5(7):665–670, 2002.

[12] D.H. Hubel and T.N. Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574, 1959.

[13] D.H. Hubel and T.N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106, 1962.

[14] T. Kadir and M. Brady. Saliency, scale and image description. *International Journal of Computer Vision*, 45(2):83–105, 2001.

[15] T.S. Lee and D. Mumford. Hierarchical Bayesian inference in the visual cortex. *JOSA A*, 20(7):1434–1448, 2003.

[16] B. Leibe and B. Schiele. Scale-invariant object categorization using a scale-adaptive mean-shift search. *Pattern Recognition*, pages 145–153, 2004.

[17] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[18] S. Marsland. *Machine learning: an algorithmic perspective*. Chapman & Hall/CRC, 2009.

[19] S.B. McGrayne. *The Theory That Would Not Die: How Bayes' Rule Cracked the Enigma Code, Hunted Down Russian Submarines, and Emerged Triumphant from Two Centuries of Controversy*. Yale Univ Pr, 2011.

[20] V.B. Mountcastle. Modality and topographic properties of single neurons of cat's somatic sensory cortex. *J Neurophysiol*, 20(4):408–434, 1957.

[21] V.B. Mountcastle. The columnar organization of the neocortex. *Brain*, 120(4):701, 1997.

[22] H. Neumann and W. Sepp. Recurrent V1–V2 interaction in early visual boundary processing. *Biological Cybernetics*, 81(5):425–444, 1999.

[23] National Academy of Engineering. Reverse-engineer the brain, 2005.

[24] GA Orban, L. Lagae, A. Verri, S. Raiguel, D. Xiao, H. Maes, and V. Torre. First-order analysis of optical flow in monkey brain. *Proceedings of the National Academy of Sciences of the United States of America*, 89(7):2595, 1992.

[25] L. Rabiner and B. Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[26] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[27] M. Riesenhuber and T. Poggio. Hierarchical models of object recognition in cortex. *nature neuroscience*, 2:1019–1025, 1999.

[28] G. Roth and U. Dicke. Evolution of the brain and intelligence. *Trends in Cognitive Sciences*, 9(5):250–257, 2005.

[29] T. Serre and M. Riesenhuber. Realistic modeling of simple and complex cell tuning in the hmaxmodel, and implications for invariant object recognition in cortex. 2004.

[30] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio. Robust object recognition with cortex-like mechanisms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(3):411–426, 2007.

[31] K. Tanaka. Inferotemporal cortex and object vision. *Annual review of neuroscience*, 19(1):109–139, 1996.

[32] T. Teichert, T. Wachtler, F. Michler, A. Gail, and R. Eckhorn. Scale-invariance of receptive field properties in primary visual cortex. *BMC neuroscience*, 8(1):38, 2007.

[33] A.M. Turing. Can digital computers think? *The Turing test: verbal behavior as the hallmark of intelligence*, page 111, 2004.

[34] D.C. Van Essen and J.L. Gallant. Neural mechanisms of form and motion processing in the primate visual system. *NEURON-CAMBRIDGE MA-*, 13:1–1, 1994.

[35] J. Von Neumann, P.M. Churchland, and P.S. Churchland. *The computer and the brain*. Yale Univ Pr, 2000.

[36] A.P. Witkin. Scale-space filtering. *Readings in computer vision: issues, problems, principles, and paradigms*, pages 329–332, 1987.

Appendix A

Software

This appendix provides an outline of the software developed for this paper. There were two components developed. One that implements the HMAX object recognition fitler, and one that implements HTM Z1.

A.1 ORFilter

This class defines the Object Recognition Filter. It applies the filter to the current image. The mode defines which filter to use.

Parameters

- Pointer to current image.
- Filter mode.

Three filters are possible and are constructed using modular HMAX layers.

1. Layer S1.
2. Layer S1 and S2.
3. Full HMAX implementation with S1, C1, S2, and C2 layers.

A.1.1 Filter Layers

There are 4 filter layers which are the heart of HMAX. All layers perform thier respective filtering function centered over a region of interest or *receptive field* in the previous hierarchical layer. In the case of the first layer, the ROI projects onto the raw incoming image.

- **Input Layer**. This is the input image.
- **NDPFilter**. Implements a Gabor Filter hierarchical layer S1. This is duplicated for as many Gabor rotations as required.
- **MaxFilter**. Max Filter for layer C1. Performs Maximum winner-take-all function on input data.
- **GRBFFilter**. RBF Filter for layer S2. Performs feature matching against a generic feature database.
- **GMaxFilter**. Max Filter for layer C2. Performs a Maximum winner-take-all function that finds the magnitude of maximum response.

A.1.2 Auxiliary Procedures

- **GetNextImage**. Copies the new image into memory. Creates the image pyramid.
- **CalcSiParams**. Calculates the receptive fields for the input image layer.
- **CalcParams**. Calculates the receptive fields for all remaining filter layers.

A.2 HTMZ1

This class implements Hierarchical Temporal Memory (version zeta 1). A 5 layer model is created using modular computing nodes. The number of nodes in each layer is shown in Table A.1.

Public Methods

	size
Layer 0	1
Layer 1	2x2
Layer 2	4x4
Layer 3	8x8
Layer 4	16x16

Table A.1: HTM Hierarchical Layers

- **Train(size)**. Trains the HTM network using a image sequence with **size** frames.
- **Super(class)**. Sets the supervised learning to **class**.
- **Group(layer)**. Performs temporal grouping in an HTM **layer** on learned sequences in the Transition Adjacency Matrix.
- **Run(image)**. Runs fully trained HTM network using the input **image**;
- **Save()**. Saves a trained HTM network to disk.
- **Restore()**. Restores an HTM network from a previously saved file.

A.2.1 Node

This class defines the HTM computing node. Each Node instantiates both a Temporal and Spatial Pooler, connecting them in series with the Spatial Pooler feeding into the Temporal Pooler.

Private Methods

- **GetChildData()**. Sends child messages up to the next hierarchical layer.
- **GetInputData()**. Copies input data into the first layer.

Public Methods

- **Run()**. Runs fully trained HTM node on input data.
- **Save()**. Saves all quantization centers and temporal groups.
- **Restore()**. Saves all quantization centers and temporal groups.

A.2.2 TPool

This class defines the Temporal Pooler which groups temporally adjacent inputs from the Spatial Pooler by determining which sequences are highly connected with others.

Private Methods

- **Learn()**. Update the Transition Adjacency Matrix.
- **Map()**. Maps Temporal Pooler input to an existing temporal group.
- **MakeGroup()**. Creates temporal groups.

Public Methods

- **Run(mode)**. If **mode** is zero no-op. If mode is one, execute the Learn() procedure. If mode is two, execute the Map() procedure.
- **Save()**. Saves all temporal groups to a file.
- **Restore()**. Restores all temporal groups from a file.

A.2.3 SPool

This class defines the Spatial Pooler, which performs a Vector Quantization like operation on given input data. The input data is drawn from a non-overlapping region of interest which is defined for each computing node.

Private Methods

- **DistM()**. Compute the Manhattan distance between two vectors.
- **AddCenter()**. Add a quantization center.
- **Learn()**. Adds the current input vector to the code book if it is outside a threshold value to any of the code book entries.
- **Map()**. Returns the index of the code book entry that is closest to the current input vector.

Public Methods

- **Run(mode)**. If **mode** is zero nop. If mode is one, execute the Learn() procedure. If mode is two, execute the Map() procedure.
- **Save()**. Saves all quantization centers to a file.
- **Restore()**. Restores all quantization centers from a file.

A.2.4 Link

The Link class defines the Image and Control transport layer for the SRV1 Robot.

Private Methods

- **SetVideoMode()**. Sets the image stream resolution.
- **CheckSockets()**. Returns the number of open TCP/IP sockets.
- **SocketRdy()**. Waits until a socket is ready.
- **Getframe()**. Gets the next image frame from the robot.
- **StartOfFrame()**. Detects the start of an image frame.
- **EndOfFrame()**. Detects the end of an image frame

Public Methods

- **TCPSend()**. Send a packet of information to the robot.
- **TCPRecv()**. Recieve a packet of information from the robot.
- **ShowFrame()**. Display the current image on the screen.
- **InitLink()**. Initialize the robotic link.
- **QuitLink()**. End the robotic link.
- **Laser()**. Turn robot lazers off and on.
- **Drive(direction)**. Commands the robot to drive forward, backward, left or right.
- **MotorsOn()**. Turn on robot drive motors.
- **Speed(speed)**. Sets the robot's speed to fast or slow.
- **SlowForward()**. Command robot to drive forward very slowly.